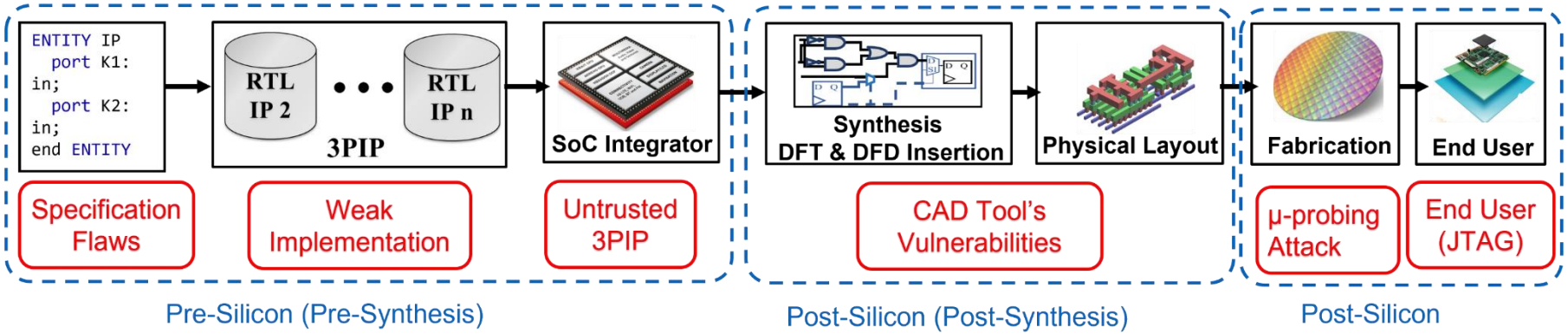
SoCureLLM: An LLM-driven Approach for Large-Scale System-on-Chip Security Verification and Policy Generation

Shams Tarek

Ph.D. Supervisor: Dr. Farimah Farahmandi

Security & Trust Issues: Supply Chain

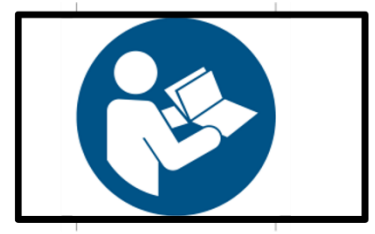


- **3PIP providers**
  - Working under aggressive schedules → **design mistakes, poor IP validation**
  - Can insert malicious implants (hardware **Trojans**)

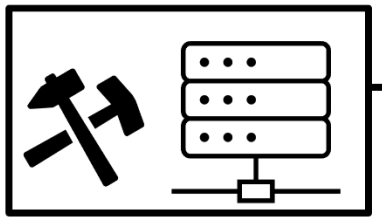
- **CAD tools**
  - **Not equipped** with understanding security vulnerabilities
  - Vulnerabilities during **optimization, synthesis, DFT**, etc.

- **Foundry**
  - Access to the entire design → hardware Trojan, Counterfeit
  - **Counterfeits** → low-quality clones, overproduced chips in untrusted foundry

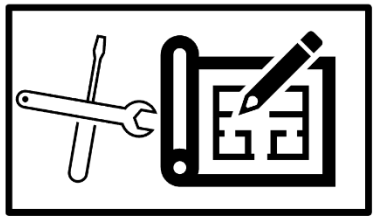
Traditional Security Verification Approaches



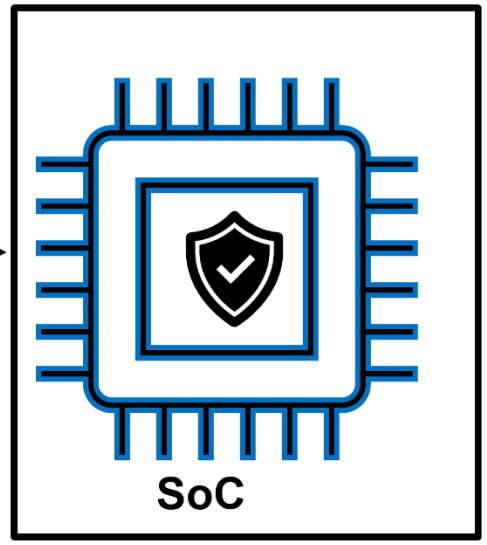
Manual Effort



Mining Simulation Traces

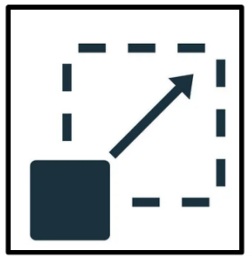


Design Instrumentation



Security Verification on SoC

Time Consuming



Scalability Issue

Erroneous

No Golden Benchmarks

Limitations

- Incomplete Verification
- Less trustworthiness

Motivation

Existing solutions to SoC security verification

- based on manual, ad-hoc strategies and limited sets of security properties and rules
- struggle with scalability, comprehensiveness, and adaptability
- come short in database creation, benchmarking, and countermeasure development

Verification Method	Scalability	Adaptability	Automation
Assertion-based Verification (Formal)	Moderate	Low	Low
Information Flow Tracking	Very Low	Moderate	High
Fuzzing and Penetration Test	High	Low	High
Simulation	Low	Very Low	Low

Why LLM in SoC Security Verification?

- Natural language understanding, advanced reasoning, and knowledge transfer capabilities
- Analyze design documentation and codebases to learn the specific characteristics of one design
- Apply learned knowledge to another design potentially revealing security weaknesses that are not immediately apparent
- Automate parts of the verification process, significantly reducing the need for manual analysis

https://arxiv.org/abs/2310.06046

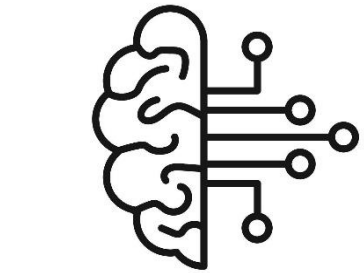
Challenges in SoC Security Analysis with LLM

Large SoC Design

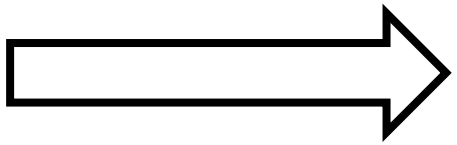
```
module csr_regfile import ariane_pkg::*; #(
    parameter logic [63:0] DmBaseAddress   = 64'h0, // debug
    parameter int          AsidWidth       = 1,
    parameter int unsigned NrCommitPorts   = 2,
    parameter int unsigned NrPMPEntries    = 8
) (
    input  logic                  clk_i,
    input  logic                  rst_ni,
    input  logic                  time_irq_i,
    // send a flush request out if a CSR with a side effect h
    output logic                  flush_o,
    output logic                  halt_csr_o,
    // commit acknowledge
    input  scoreboard_entry_t [NrCommitPorts-1:0] commit_inst
    input  logic [NrCommitPorts-1:0]              commit_ack_
    // Core and Cluster ID
    input  logic[riscv::VLEN-1:0] boot_addr_i,
    input  logic[riscv::XLEN-1:0] hart_id_i,
    // we are taking an exception
    input exception_t             ex_i,

    input  fu_op                  csr_op_i,
    input  logic  [11:0]          csr_addr_i,
    input  logic[riscv::XLEN-1:0] csr_wdata_i,
    output logic[riscv::XLEN-1:0] csr_rdata_o,
    input  logic                  dirty_fp_state_i,
    input  logic                  csr_write_fflags_i,
    input  logic  [riscv::VLEN-1:0]  pc_i,
    output exception_t            csr_exception_o,

    // Interrupts/Exceptions
    output logic  [riscv::VLEN-1:0] epc_o,
    output logic                  eret_o,
    output logic  [riscv::VLEN-1:0] trap_vector_base_o,
    output riscv::priv_lvl_t      priv_lvl_o,
    // FPU
    output riscv::xs_t            fs_o,
    output logic [4:0]            fflags_o,
    output logic [2:0]            frm_o,
    output logic [6:0]            fprec_o,
    // Decoder
```



LLM



Most of the open-source LLM fail to analyze large Verilog modules because of token limitation

Security reviews are not good enough

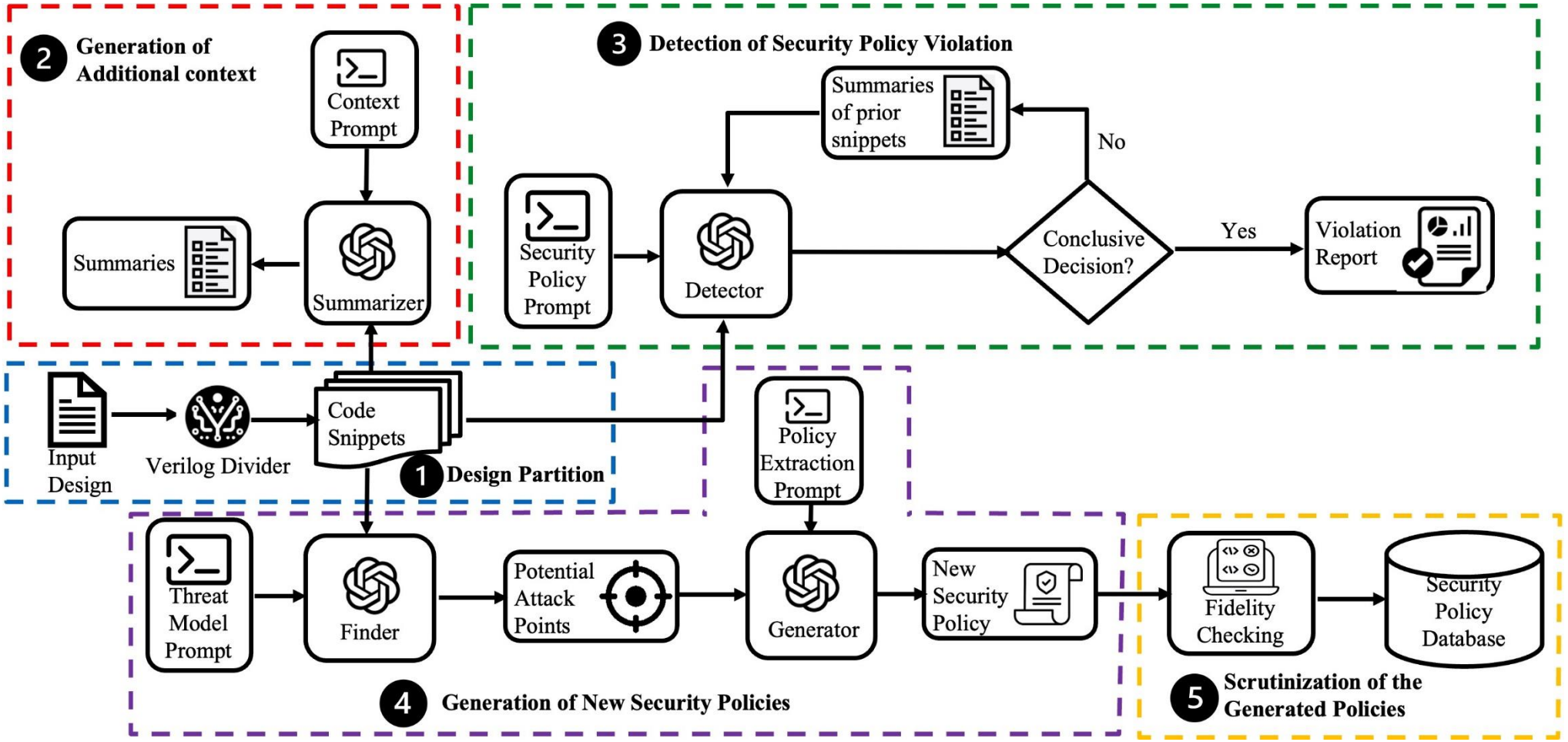
LLMs fail to perform fine-grained security analysis

Contribution

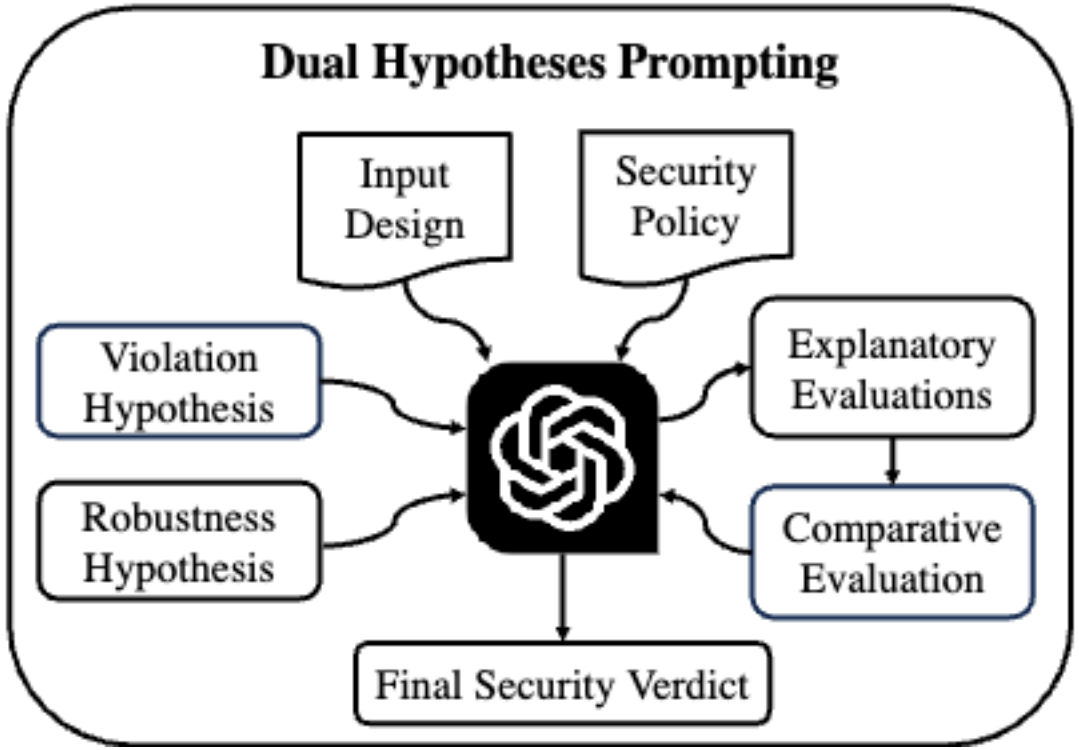
- Introduces the first LLM-based framework for security verification of large-scale SoC designs.
- Overcomes LLM limitations such as token constraints and context memorization through modular processing.
- Enables end-to-end identification of security bugs across complex RTL structures.
- Builds a comprehensive security policy database from threat model-driven analysis.
- Demonstrates superior performance on buggy SoC benchmarks compared to existing verification techniques.

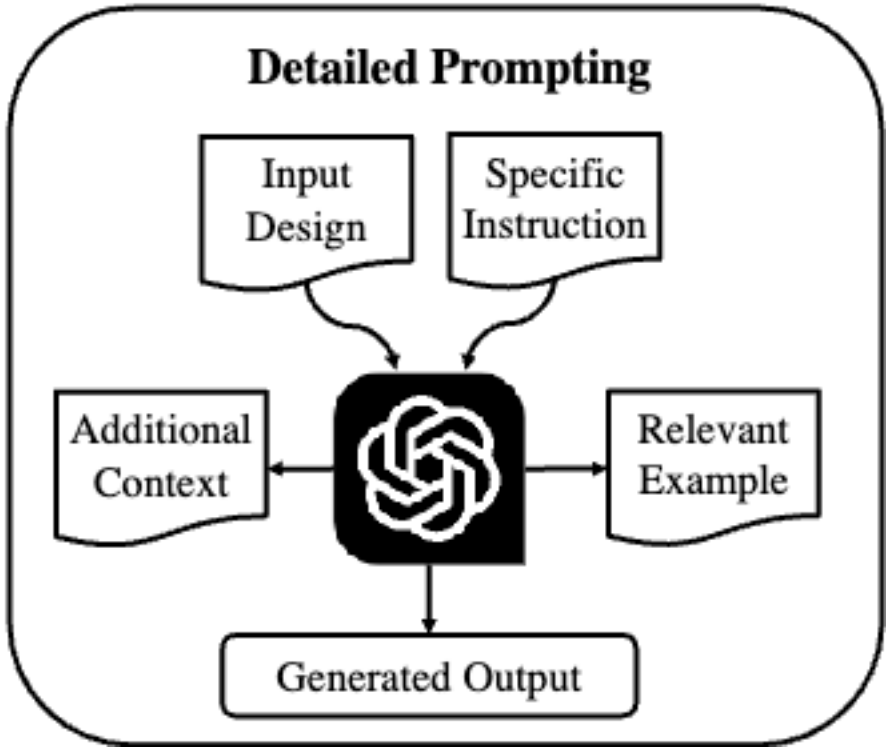
SoCureLLM Framework

- An LLM-driven Approach for Large-Scale System-on-Chip Security Verification and Policy Generation



Prompting Strategies





List of Security Bugs

Vuln.	Vuln. Name with CWE ID
V1	CWE-1198: Improper handling of privilege issues
V2	CWE-269: Improper privilege level during interrupt handling
V3	CWE-1245: Incorrect FSM transition logic
V4	CWE-1260: Overlapping between memory ranges
V5	CWE 190- Integer Overflow or Wraparound
V6	CWE-506: Hardware trojan inside the decoder module
V7	CWE-310: Trojan in AES for information leakage
V8	CWE-310: Trojan in AES for denial of service
V9	CWE-1244: Unlocking JTAG during reset
V10	CWE-310: Trojan in CSR module unauthorized access
V11	CWE-321: Use of hardcoded cryptographic key
V12	CWE-250: Improper trap privilege assignment
V13	CWE-284: Improper direct memory access
V14	CWE-1271: Unauthorized access to important registers

Case Study

Dual Hypothesis Prompting (C.1)

//Input Code Snippet & Summaries will be added to the prompt automatically//

Security Policy: The debug module's password-checking mechanism must be logically correct, and all related signals should be cleared during the reset state.

Consider two scenarios: First, assume that the security policy is breached in the provided code snippet and the technical summaries. Write an analysis based on this assumption.
Second, assume that the security policy is adhered to in the provided code snippet and the technical summaries. Write an analysis based on this assumption.

Compare the two analyses and identify the more logical one. If you reach to a conclusive decision, provide a detailed explanation based on your analysis. If your decision is inconclusive, you will request additional summaries from the previous snippets sequentially.

Your response should strictly maintain the following structure:

Analysis under Breach of Security Policy Assumption:
add your response here

Analysis under Adherence to Security Policy Assumption:
add your response here

Final Verdict:
add your response here
"The decision is conclusive" or "The decision is inconclusive"

Response from GPT-4 without correct Summary (C.2)

Analysis under Breach of Security Policy Assumption:
Retention of previous state information (`passchk`, `correct`, `bitmask`) in other non-reset states can be exploited if these variables are not appropriately re-initialized when expected. An attacker could manipulate the transition conditions by ensuring that `correct` hits the specific value, thereby setting `passchk` to 1 without a legitimate authentication sequence.

Analysis under Adherence to Security Policy Assumption:
If the system enters the reset state, it should explicitly clear signals like `passchk`, `correct`, `bitmask`, and others to adhere to the policy that all related signals should be reset. This ensures that any remaining or leftover signal values from prior operations that could jeopardize security are eliminated each time the machine is reset.

Final Verdict:
A definitive conclusion on vulnerability cannot be reached without further details on other parts of the system. Additional snippets or a broader view of the full implementation would be necessary to determine compliance or breach of the stated security policy conclusively.

The decision is inconclusive.

Response from GPT-4 with correct Summary (C.3)

Analysis under Breach of Security Policy Assumption:
Because `correct` is not cleared, an attacker who successfully unlocks the JTAG module once can bypass the password-checking mechanism in subsequent attempts. The counter value remains above the conditional check (`correct == 32'h8000_FFFF`), automatically setting `passchk` to 1, which should only happen after a successful password check.
This persistent `correct` signal effectively disables the password-checking mechanism after one successful entry, making the module vulnerable to unauthorized access without further password verification. The module stays unlocked indefinitely, a severe security risk allowing persistent, unauthorized access.

Analysis under Adherence to Security Policy Assumption:
With the additional context provided, it becomes evident that despite resetting `passchk`, failing to reset `correct` is a significant oversight. This non-compliance with the security policy undermines any other security measures implemented in the code.
The initial analysis assumed the other signals might be correctly managed based on the snippet. However, knowing now that `correct` is not reset, it is clear that the implementation does not adhere to the essential security policy of clearing all sensitive signals during a reset.

Final Verdict:
Based on the combined analysis of both code snippets and the new summary provided, the code is definitively vulnerable. The failure to reset the `correct` signal allows for an easily exploitable loophole that can leave the module unlocked indefinitely, providing unauthorized access without subsequent password verification. This represents a serious security flaw requiring immediate correction to comply with security policies and ensure the integrity of the module.

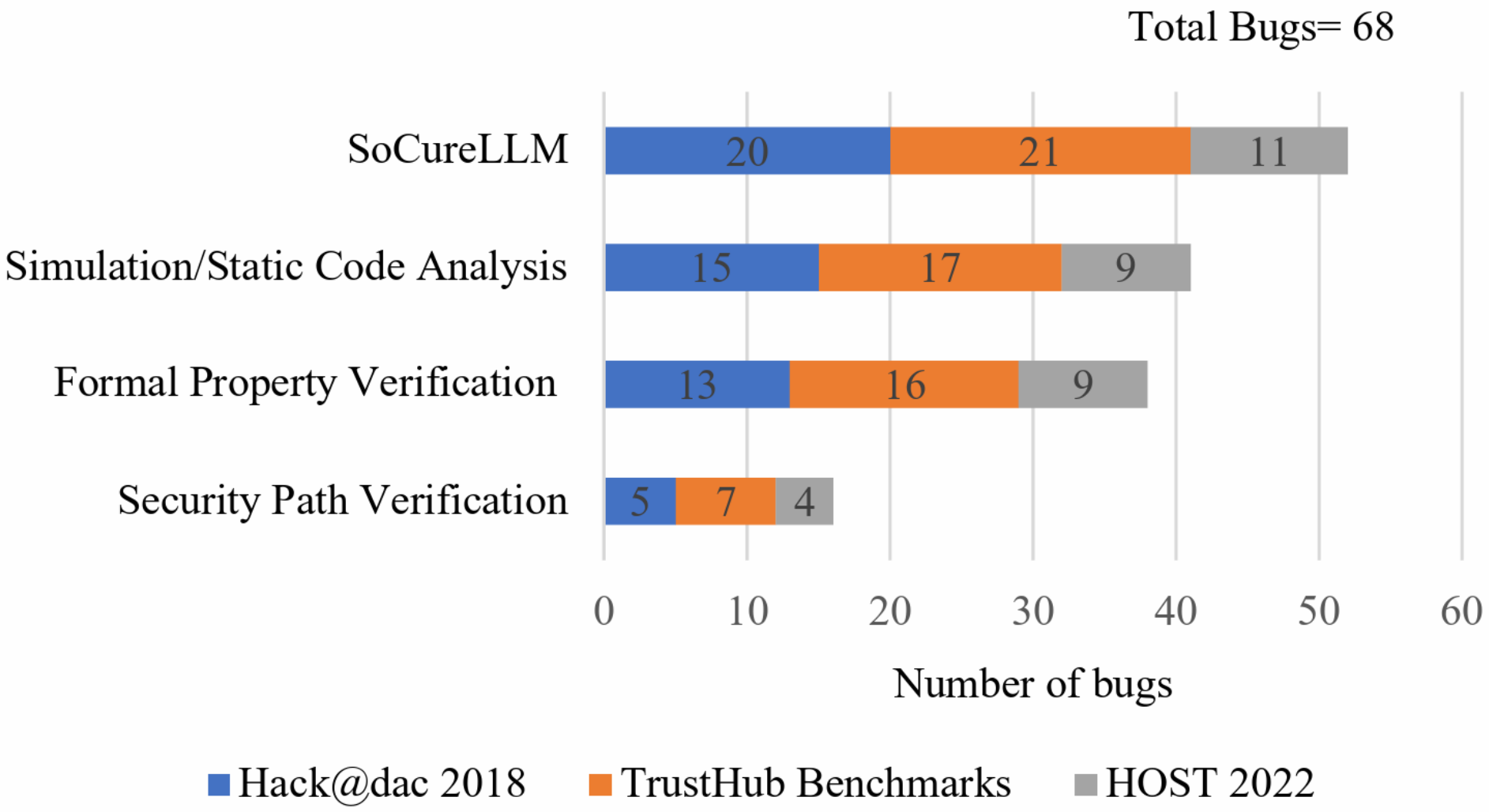
The decision is conclusive.

SoCureLLM: Experimental Results

Table 2: Performance comparison between open-ended security assessment and proposed SoCureLLM in the detection of security bugs

Design	IP	LoC	Open Ended Security Assessment					Security Policy Violation Assessment (SoCureLLM)			
			# bugs in the module	# bugs detected	# bugs detected successfully	TPR	TNR	# bugs detected	# bugs detected successfully	TPR	TNR
Hack@Dac2018	Debug Unit	715	7	5	4	0.571	0.950	8	6	0.857	0.900
	GPIO	408	5	3	2	0.400	0.955	4	4	0.800	1.000
	CSR	1510	3	1	0	0.000	0.958	2	2	0.667	1.000
	RISC-V Core	14635	7	3	2	0.286	0.950	9	5	0.713	0.800
	AXI Interface	810	1	0	0	0.000	1.000	1	1	1.000	1.000
	Crypto Modules	11606	4	2	1	0.250	0.957	2	2	0.500	1.000
	Total	**29684**	**27**	**14**	**9**	**0.333**	**0.963**	**26**	**20**	**0.741**	**0.956**
Trust-Hub Benchmark	CSR	1510	10	4	3	0.300	0.944	8	8	0.800	1.000
	RISC-V Core	14635	4	5	2	0.500	0.870	5	4	1.000	0.957
	Decoder	1418	4	2	1	0.25	0.957	4	3	0.750	0.957
	MMU	519	2	0	0	0.000	1.000	1	1	0.500	1.000
	PMP	278	1	0	0	0.000	1.000	1	1	1.000	1.000
	AES	12624	4	2	2	0.500	1.000	3	3	0.750	1.000
	AXI Interface	810	2	1	1	0.500	1.000	2	1	0.500	0.960
	Total	**31794**	**27**	**14**	**9**	**0.333**	**0.970**	**24**	**21**	**0.778**	**0.982**
HOST 2022	CSR	1510	4	3	2	0.500	0.900	3	3	0.750	1.000
	Crypto Module	8133	5	2	2	0.400	1.000	5	4	0.800	0.890
	RISC-V Core	14635	4	3	2	0.500	0.900	5	3	0.750	0.800
	memory unit	235	1	0	0	0.000	1.000	1	1	1.000	1.000
	Total	**24513**	**14**	**8**	**6**	**0.423**	**0.952**	**14**	**11**	**0.786**	**0.923**

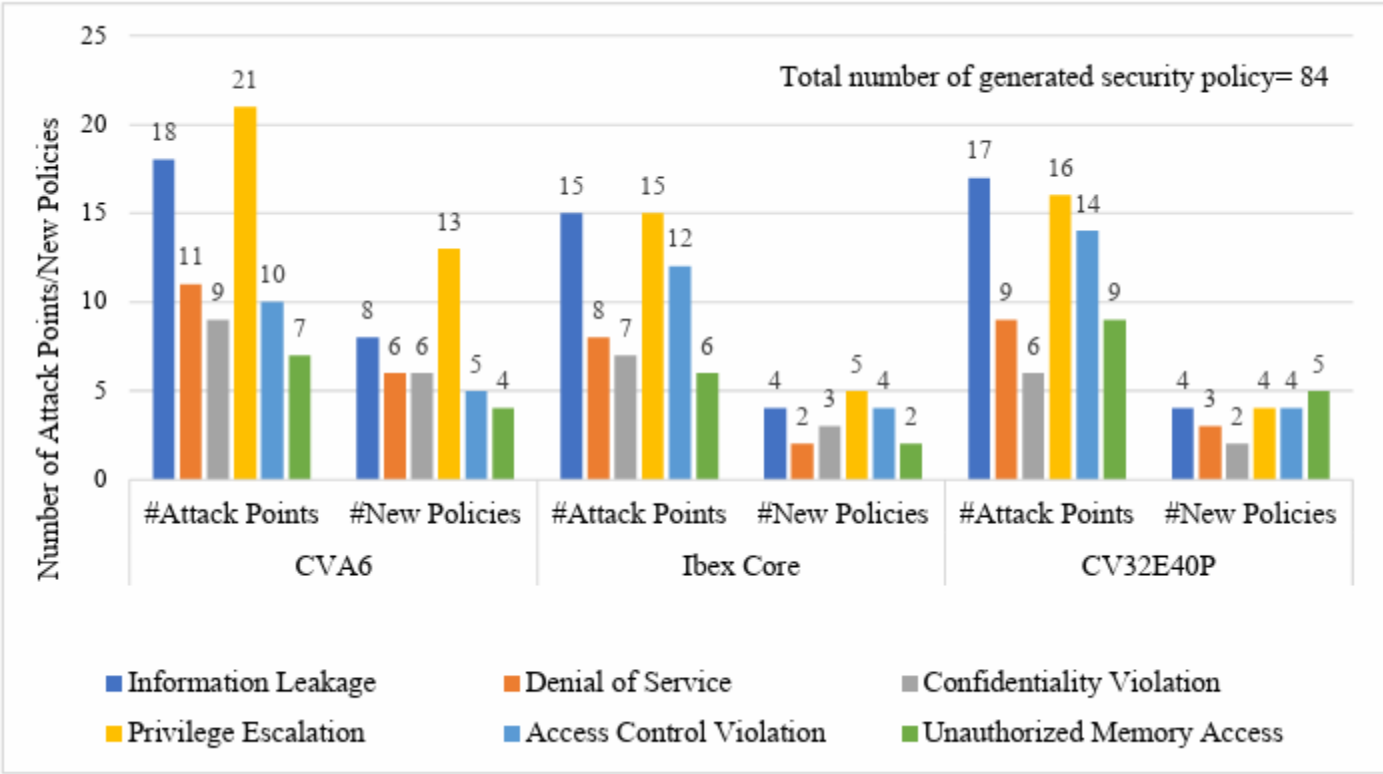
SoCureLLM: Experimental Results



SoCureLLM: Policy Generation

Sample of generated security policies

- Monitor the 'debug_mode_o' signal to ensure it does not leak through side channels.
- Review state transitions for potential escalation vectors, especially around 'current_priv_lvl_i'.
- Implement rate-limiting and sanity checks on the 'irq_req_ctrl_i' signal to prevent IRQ flooding.
- Implement strict access control checks on debug and control registers.
- Audit the control flow for any unauthorized bypasses or weak checks.
- Implement bounds checking for memory accesses and handle exceptions for 'data_misaligned_i'.



Key Takeaways

- **Automated and Adaptable Implementation:** SoCureLLM simplifies security validation by using consistent prompting strategies, reducing manual intervention and easily adapting to new vulnerabilities without structural changes.
- **Scalability for Large Designs:** The framework uses rule-based partitioning and iterative decomposition to effectively handle complex, large-scale SoC designs.
- **LLM-Agnostic Compatibility:** Designed to work with both open-source and proprietary LLMs, SoCureLLM ensures platform flexibility and broader applicability.
- **Controlled Output Variability:** By using a consistently low temperature setting, the framework ensures deterministic LLM responses.

Thank You!

Questions?